25. (Amended) The computer program product as claimed in claim 23, further comprising changing the status of the object to global when the monitoring step determines that the object is accessible from either of a global root or other global object.

26. (Amended) The computer program product as claimed in claim 23 wherein the status of an object in the thread heap is changed to global if the object is assigned to a static variable or if the object is assigned to a field in a global object.

REMARKS/ARGUMENTS

This communication is responsive to a non-final Office Action dated August 29, 2002 rejecting claims 1-26 of the instant patent application (hereafter the "Office Action"). Therefore, claims 1-26 have been amended. Reconsideration of the application is respectfully requested.

I. Drawings

The Office Action objected to the drawings under 37 CFR § 1.83(a) for failing to show step 4.8 of Figure 4. Accordingly, a copy of Figure 4 showing a proposed amendment correcting the error is attached. Therefore, the objection to the drawings has been overcome.

II. Claim Rejections – 35 USC § 112

The Office Action rejected claim 6 under 35 USC § 112, second paragraph as being indefinite on grounds that there is insufficient antecedent basis for "a thread heap." Claim 6 has been amended to provide proper antecedent basis.

III. Claim Rejections – 35 USC §102(b)

Claims 1-7, 14, 16, 17, and 22-26 have been rejected under 35 U.S.C. §102(b) as being

anticipated by Kolodner et al. (US Pat. No. 6,289,360). This rejection is overcome because it is believed that claims 1-7, 14, 16, 17, and 22-26, as amended, are not anticipated by Kolodner. Nowhere does Kolodner teach or disclose monitoring objects as required by claims 1-5. Instead, Kolodner discusses tracing objects from the stack and then tracing references from within those objects.

Kolodner does not discuss monitoring each object that is local to the given thread to determine whether the object is accessible from any thread other than the given thread. Kolodner does not even mention the possibility of tracing based on the local/global nature of an object. Moreover, Kolodner does not distinguish the objects accessible from the thread stacks from those reachable from other roots. Therefore, Kolodner neither teaches nor suggests the invention of claim 1. The claims dependent on claim 1 are distinguishable from Kolodner at least for the same reasons that claim 1 is distinguishable therefrom.

Claim 2 now requires assigning a status to the given object wherein the status designates the object as a local object. Kolodner at col. 9, lines 28-36 discusses a status variable but that variable does not relate to the local or global status of an object (rather Kolodner discusses local and global states of *threads*).

Claim 3 recites the step of deleting from the thread heap one or more local objects when it is determined that they are not accessible from a local root. The Office Action cited col. 2, lines 8-11 of Kolodner. That section discusses the determination of whether objects are "live." It does not relate to deletion of objects that are determined to be not accessible from a local root.

Claim 4, as amended, requires changing the object status to global when it is determined that the object is accessible from either a global root or another global object. The Office Action cited col. 8, lines 40-53 and col. 11, lines 46-47 of Kolodner. As stated above, Kolodner does not teach or suggest monitoring the local/global status of objects.

Kolodner does not anticipate claims 5-7, 14, 16, 17, and 22-26 for at least the same reasons as discussed above.

IV. Claim Rejections – 35 USC §103(a)

The Office Action rejected claims 8, 9, and 15 as obvious over the Kolodner '360 patent in view of US Patent 5,966,702 to Fresko.

The Fresko patent deals with grouping class files together in single combined multi-class file (called a jar file in Java terminology). In particular the patent deals with the pre-processing and packaging of the class files. The Kolodner patent deals with a mechanism for reducing/eliminating the synchronization between sweep and allocation for a concurrent garbage collector.

Each of the elements recited in claims 8, 9, and 15 are claimed in combination with their base claim (or claims) and would not have been obvious to those skilled in the art because of the merits of the combinations set forth therein. Moreover, the teachings of Fresko do not relate to the problem of marking objects for collection at all. Therefore, Applicant respectfully contends the combination of Fresko with Kolodner falls far short of the invention claimed in claims 8, 9, and 15 In view of the remarks made in support of the base claims, the additional elements recited in claims 8, 9, and 15 are patentable in combination with the base claims.

The Office Action rejected claims 10-13 as obvious over Kolodner in view of Fresko and further in view of O'Connor et al (US Patent No. 5953736). Regarding claim 10, O'Connor discusses a 32-bit word with additional storage reserved for synchronization status of the object. However, O'Connor neither teaches nor suggests the use of spare capacity in an object header for status of the object.

Regarding claim 11, as discussed above with respect to claim 10, O'Connor does not discuss using the spare space available in the header portion as the flag, or status of the object. Moreover, since O'Connor does not at all relate to monitoring the local/global status of an object, there is no suggestion, motivation, or teaching that would have motivated one skilled in the art to combine the references.

Regarding claim 12, Fresko does not disclose in its Abstract moving objects whose status is global from the thread heap to a global heap. In fact, Fresko does not discuss global objects or

global heaps.

The Office Action rejected claims 18-21 as being obvious over Kolodner in view of Jagannathan et al. (US Patent No. 5,692,193). Jagannathan relates to software architecture for control of highly parallel computer systems. Jagannathan is cited for teaching the local thread stacks and heaps and a global heap. Applicant respectfully submits that the claimed respective local thread stacks and heaps, and a global heap must be viewed as part of the claimed combination and that absent some evidence of a teaching, suggestion or motivation the mere presence in a prior reference of elements having similar or same names is not a proper basis for a conclusion of obviousness. Moreover, even if they were properly combinable, the cited references still fail to teach or suggest the claimed means for monitoring each object that is local to the given thread, to determine whether the object is accessible from any thread other than the given thread. Claims 18-26 are also patentable for the reasons discussed above with respect to the claims dependent on claim 1 which is a method counterpart to independent claims 18 and 22. Therefore, claims 18-26 would not have been obvious to those skilled in art in view of the cited references.

Attached hereto is a marked-up version of the changes made to the specifications and claims by the current amendment. The attached page is captioned **"Version with markings to show changes made."**

Applicant respectfully requests that a timely Notice of Allowance be issued in this case.

Respectfully submitted,

Please send correspondence to:
Michael J. Buchenhorner, P.A.
1430 Sorolla Avenue
Coral Gables, Florida 33134

By: Michael J. Buchenhorner
Michael J. Buchenhorner
Registration No. 33,162
Telephone No.: (305) 529-0221

VERSION WITH MARKINGS TO SHOW CHANGES MADE

In the specification:

Please replace the paragraph beginning at page 2, line 29 of the specification with the following:

--Many current implementations of Java use the classic mark-sweep-compact method of garbage collection as delivered in the base SUN JVM. References to the objects that are being processed at any instant by the system are stored in the registers, one or more thread stacks and some global variables. The totality of objects that [are maybe] may be needed by the system can be found by tracing through the objects directly referenced in the registers, stacks, and global variables and then tracing through these "root" objects for further references. The objects in use by a system thereby form a graph rooted at the root objects and any extraneous objects are not part of this graph. Once all the objects in the graph are found, the remaining objects in the heaps may be discarded (garbage collected).--

Please replace the paragraph beginning at page 8, line 15 of the specification with the following:

--Referring to Figure 1, there is shown a computer platform 10 such as [Pentium] a Pentium processor based PC typically comprising a motherboard with processor, a 32Mbytes RAM memory, and interface circuitry; a 4G Byte hard drive; a 20x CD ROM; an SVGA monitor; a keyboard and a mouse. On [power on] power-on the platform loads into the memory an operating system 12 such as Windows NT v4 and subsequently a Java Virtual Machine (JVM) based on the Java Development Kit (JDK) v1.1 or v1.2. The Java virtual machine is capable of loading a Java application 16 from disk into memory so that the application may be executed on the platform 10. The Java application 16 is object orientated program code and creates Java language objects 18A, 18B, 18C, and 18D. An example of how the objects are built by the JVM is described with reference to Figure 3.--

Please replace the paragraph beginning at page 9, line 8 of the specification with the following:

--The runtime data area 26 comprises thread memory space 28A, 28B, 28n, global heap 34, class area 36 and compiled method area 38. The thread memory space 28 ideally stores local thread objects, local thread execution data and also objects marked global. Local objects are objects which are used by one thread alone and are not common between. Thread objects are stored in thread heaps 32A, 32B, 32n and thread execution data including object references and register data are stored in the thread stacks 30A, 30B, 30n. The global heap 34 stores objects which are common to more than one thread, that is, global objects. A local object which is loaded into a thread heap 32A for processing by thread 28A may become global if it is later used by another thread, say thread 28B, and may remain in the thread heap 28A or be moved to the global heap 34 by 'stop everything'. Class area 36 stores the classes as they are loaded into the JVM, classes are defined as global because they are common to all the threads hence any field of a class is globally reachable. An object, which is an instance of a class, is placed in a thread heap and is initially local because it may only be used by that thread. An exception is instance objects of class Thread or Runnable which are global at creation because they can be reached by other threads. Whereas an object referenced by a variable in a class is global because the object is reachable from the class which is reachable by all threads. The compiled method area 38 is used by the JIT compiler 24 to store native executable code compiled from the Java byte code.--

Please replace the paragraph beginning at page 13, line 8 of the specification with the following:

--The process of memory management is described with reference to Figure 4. Step 4.1 creates an object [is created] from a class stored in the class area 36. Step 4.2 [check] checks to see if the class is a 'global' class, i.e., a class all of whose instances are global or one whose instances we expect to quickly become global, whereby the object is assigned global and/or placed in the global heap (Step 4.7). Step 4.3 calculates the size of the object [is calculated] to see whether it will fit in the thread heap. Step 4.4 performs garbage collection to free up memory

if there is not enough space [then garbage collection is performed to free up memory]. Step 4.5 [place] places the object in the thread heap memory along with the object length in multiples of eight. Step 4.6 [Use] uses a spare bit in the length attribute as a flag and set as local ('0'). The process ends at step 4.8.--

Please replace the paragraph beginning at page 13, line 22 of the specification with the following:

--The write operation and write barrier process are described with reference to Figure 5. In Step 5.1 a write operation is called and intercepted by the write barrier which is integrated with the write operation code. Step 5.2 [Check] checks whether the object is being assigned to a static variable of a class and if so set the status of the object as global (step 5.4) and set all objects reachable by the object as global (step 5.5) before moving to step 5.6, if not so then do step 5.3. Step 5.3 [Check] checks whether the write operation assigns the object to another global object, if so set the status of the object as global (step 5.4) and set all objects reachable by the object as global (step 5.5). Once complete continue with the write operation.--

In the claims:

1. (Amended) A method of managing memory in a multi-threaded processing environment including [respective] local thread stacks and local thread heaps, and a global heap, said method comprising:

creating an object in a thread heap; and

for a given thread, monitoring each object that is local to the given thread, to determine whether the object is [reachable] accessible from any thread [anywhere] other than the given thread [stack].

2. (Amended) [A] The method as claimed in claim 1 further comprising:

[associating a local] assigning a status [with] to the given object, the status designating the object as a local object[;]

[changing the status of the object to global under certain conditions].

3. (Amended) [A] The method as claimed in claim 2 further comprising deleting from the thread heap one or more local objects when it is determined that they are not [reachable] accessible from a local root.

4. (Amended) [A] The method as claimed in claim [3] 2, further comprising changing the status of the object to global when the monitoring step determines that the object is accessible from either of a global root or other global object [where reachability is determined by tracing from the local root].

5. (Amended) [A] The method as claimed in claim [4 wherein the] 2, further comprising changing the status of an object in the thread heap [is changed] to global if the object is assigned to a static variable or if the object is assigned to a field in a global object.

6. (Amended) [A] The method as claimed in claim 3 further comprising intercepting assignment operations to an object in [a] the thread heap to [access] determine whether the object status should be changed.

7. (Amended) [A] The method as claimed in claim 6 wherein the multithreaded processing environment is a virtual machine.

8. (Amended) [A] The method as claimed in claim 7 wherein the virtual machine comprises an interpreter [and the] comprising a write operation code [in the interpreter is] modified to perform [the] a checking of assignment of the object.

9. (Amended) [A] The method as claimed in claim 8 wherein the virtual machine comprises a just in time compiler wherein native compiled write operation code includes native code to perform the checking of assignment of the object.

10. (Amended) [A] The method as claimed in claim 9 further comprising using spare capacity in [the] an object header for the [flag] status.

11. (Amended) [A] The method as claimed in claim 10 further comprising using multiples of 2 or more bytes in a thread heap to store the objects whereby there is at least one spare bit in the object length variable and using the at least one spare bit as the [flag] status.

12. (Amended) [A] The method as claimed in claim 11 further comprising moving objects whose status is global from the thread heap to [a] the global heap.

13. (Amended) [A] The method as claimed in claim 12 further comprising compacting the reachable local objects in a thread heap.

14. (Amended) [A] The method as claimed in claim 1 [wherein certain objects are associated with a global status on creation] further comprising assigning a status to certain objects that designates the objects as local objects.

15. (Amended) [A] The method as claimed in claim 14 where said certain objects include class objects.

16. (Amended) [A] The method as claimed in claim 14 further comprising the step of analysing whether an object is likely to be made global and associating such an object with a global status on creation.

17. (Amended) [A] The method as claimed in claim 16 further comprising allocating objects assigned as global on creation to the global heap.

18. (Amended) A system for managing memory in a multi-threaded processing environment comprising:

respective local thread stacks and heaps;

a global heap;

means for creating an object in a thread heap; and

means for monitoring each object that is local to the given thread, to determine whether the object is [reachable] accessible from any thread [anywhere] other than the given thread [stack].

19. (Amended) [A] The system as claimed in claim 18 further comprising means for associating a local status with the object and means for changing the status of the object to global under certain conditions.

20. (Amended) [A] The system as claimed in claim 19 further comprising means for deleting from the thread heap one or more local objects when they are not [reachable] accessible from a local root.

21. (Amended) [A] The system as claimed in claim 20 further comprising:

means for changing the status of an object in the thread heap to global if the object is assigned to a static variable or if the object is assigned to a field in a global object.

22. (Amended) A computer program product stored on a computer readable storage medium for, when executed on a computer, performing a method of managing memory in a multi-threaded processing environment including [respective] local thread stacks and local thread heaps, and a global heap, said method comprising:

creating an object in a thread heap; and

for a given thread, monitoring each object that is local to the thread, to determine whether the object is [reachable] accessible from any thread [anywhere] other than the given thread [stack].

23. (Amended) [A method] The computer program product as claimed in claim 22 further comprising:

[associating a local] assigning a status with the object that designates the object as a local object[;]

[changing the status of the object to global under certain conditions].

24. (Amended) [A method] The computer program product as claimed in claim 23 further comprising deleting from the thread heap one or more local objects when it is determined that they are not reachable from a local root.

25. (Amended) [A method] The computer program product as claimed in claim [24 where reachability is determined by tracing from the local root] 23, further comprising changing the status of the object to global when the monitoring step determines that the object is accessible from either of a global root or other global object.

26. (Amended) [A method] The computer program product as claimed in claim [25] 23 wherein the status of an object in the thread heap is changed to global if the object is assigned to a static variable or if the object is assigned to a field in a global object.